UNITED STATES PATENT AND TRADEMARK OFFICE

UNITED STATES DEPARTMENT OF COMMERCE
United States Patent and Trademark Office
Address: COMMISSIONER FOR PATENTS
P.O. Box 1450
Alexandria, Virginia 22313-1450
www.uspto.gov

APPLICATION NO.	FILING DATE	FIRST NAMED INVENTOR	ATTORNEY DOCKET NO.	CONFIRMATION NO.
10/630,913	07/31/2003	Venugopal K. Srinivasamurthy	1509-437	7570

22879 7590 10/22/2009

HEWLETT-PACKARD COMPANY
Intellectual Property Administration
3404 E. Harmony Road
Mail Stop 35
FORT COLLINS, CO 80528

EXAMINER	
DAO, THUY CHAN	

ART UNIT	PAPER NUMBER
2192	

NOTIFICATION DATE	DELIVERY MODE
10/22/2009	ELECTRONIC

Please find below and/or attached an Office communication concerning this application or proceeding.

The time period for reply, if any, is set in the attached communication.

Notice of the Office communication was sent electronically on above-indicated "Notification Date" to the following e-mail address(es):

JERRY.SHORMA@HP.COM
ipa.mail@hp.com
laura.m.clark@hp.com

Office Action Summary	Application No.	Applicant(s)	
	10/630,913	SRINIVASAMURTHY ET AL.	
	Examiner	Art Unit	
	Thuy Dao	2192	

-- The MAILING DATE of this communication appears on the cover sheet with the correspondence address --

Period for Reply

A SHORTENED STATUTORY PERIOD FOR REPLY IS SET TO EXPIRE *3* MONTH(S) OR THIRTY (30) DAYS, WHICHEVER IS LONGER, FROM THE MAILING DATE OF THIS COMMUNICATION.

- Extensions of time may be available under the provisions of 37 CFR 1.136(a). In no event, however, may a reply be timely filed after SIX (6) MONTHS from the mailing date of this communication.
- If NO period for reply is specified above, the maximum statutory period will apply and will expire SIX (6) MONTHS from the mailing date of this communication.
- Failure to reply within the set or extended period for reply will, by statute, cause the application to become ABANDONED (35 U.S.C. § 133). Any reply received by the Office later than three months after the mailing date of this communication, even if timely filed, may reduce any earned patent term adjustment. See 37 CFR 1.704(b).

Status

1)☒ Responsive to communication(s) filed on *24 June 2009*.

2a)☒ This action is **FINAL**. 2b)☐ This action is non-final.

3)☐ Since this application is in condition for allowance except for formal matters, prosecution as to the merits is closed in accordance with the practice under *Ex parte Quayle*, 1935 C.D. 11, 453 O.G. 213.

Disposition of Claims

4)☒ Claim(s) *1 and 21-39* is/are pending in the application.

4a) Of the above claim(s) _____ is/are withdrawn from consideration.

5)☐ Claim(s) _____ is/are allowed.

6)☒ Claim(s) *1 and 21-39* is/are rejected.

7)☐ Claim(s) _____ is/are objected to.

8)☐ Claim(s) ____ are subject to restriction and/or election requirement.

Application Papers

9)☐ The specification is objected to by the Examiner.

10)☒ The drawing(s) filed on *07/31/03* is/are: a)☒ accepted or b)☐ objected to by the Examiner.

Applicant may not request that any objection to the drawing(s) be held in abeyance. See 37 CFR 1.85(a).

Replacement drawing sheet(s) including the correction is required if the drawing(s) is objected to. See 37 CFR 1.121(d).

11)☐ The oath or declaration is objected to by the Examiner. Note the attached Office Action or form PTO-152.

Priority under 35 U.S.C. § 119

12)☐ Acknowledgment is made of a claim for foreign priority under 35 U.S.C. § 119(a)-(d) or (f).

a)☐ All b)☐ Some * c)☐ None of:

1.☐ Certified copies of the priority documents have been received.

2.☐ Certified copies of the priority documents have been received in Application No. _____.

3.☐ Copies of the certified copies of the priority documents have been received in this National Stage application from the International Bureau (PCT Rule 17.2(a)).

* See the attached detailed Office action for a list of the certified copies not received.

Attachment(s)

1) ☐ Notice of References Cited (PTO-892)
2) ☐ Notice of Draftsperson's Patent Drawing Review (PTO-948)
3) ☐ Information Disclosure Statement(s) (PTO/SB/08) Paper No(s)/Mail Date _____.
4) ☐ Interview Summary (PTO-413) Paper No(s)/Mail Date. _____ .
5) ☐ Notice of Informal Patent Application
6) ☐ Other: _____.

DETAILED ACTION

1. This action is responsive to the amendment filed on June 24, 2009.

2. Claims 1 and 21-39 have been examined.

Response to Amendments

3. In the instant amendment, claim 1 has been amended.

4. The objection to claim 1 is withdrawn in view of Applicant's amendments.

Response to Arguments

5. Applicants' arguments have been considered.

Claim 1:

a) Limitations at issues "*augmenting a bytecode set of the virtual machine with semantically enriched opcodes, thereby constituting an application domain-specific virtual machine*" (Remark, pp. 9-12).

As an initial matter, examiner notes that the originally filed disclosure clearly defines:

> "In yet a further aspect, the invention provides for a method of optimising the performance of applications running on an interpreter-based runtime system, the method comprising augmenting the bytecode set of the interpreter with application-specific opcodes by reference to said application, thereby constituting an application domain-specific virtual machine…" (page 4, lines 8-11, augmenting/replacing a bytecode set with opcodes, emphasis added);
>
> "…This technique leads to bytecode compression by replacing every occurrence of most repetitive longest sequences by a respective macro name" (page 9, lines 28-30, augmenting/replacing a sequence of bytecode by a macro name, emphasis added); and

> "Online sEc-opcode embedding was adopted to replace the sequence of bytecode comprising the sEc-opcode with a single corresponding sEc-opcode..." (page 14, lines 3-4, augmenting/replacing a sequence of bytecode with a single opcode, emphasis added).

Accordingly, in light of the specification, "*a semantically enriched opcode*" does not exclude an opcode and/or a macro, which is augmented/replaced from a bytecode set and/or a sequence of bytecode (emphasis added).

Furthermore, in response to applicant's argument that the references fail to show certain features of applicant's invention, it is noted that the features upon which applicant relies (i.e., "...*the semantically enriched opcode recited in claim 1 must be deduced from the context of the static or dynamic behavior of a Java application*", Remarks, page 10) are not recited in the rejected claim(s). Although the claims are interpreted in light of the specification, limitations from the specification are not read into the claims. See In re Van Geuns, 988 F.2d 1181, 26 USPQ2d 1057 (Fed. Cir. 1993).

Sokolov discloses:

augmenting a bytecode set of the virtual machine with semantically enriched opcodes (e.g., FIG. 12A, conventional Java bytecode "Getfield" and "Astorex" are augmented/replaced by inventive opcode "Get_Store" (a semantically enriched opcode); FIG. 13C, conventional Java bytecode "lastore" and "dastore" are augmented/replaced by another inventive opcode "ASTOREL" (another semantically enriched opcode); FIG. 2B, col.6: 55 – col.6: 26).

thereby constituting an application domain-specific virtual machine (e.g., col.4: 3-15, col.6: 23-26, an augmented/modified Java virtual machine specific to the inventive opcodes, col.7: 51-59; FIG. 2A-B, col.8: 6 - col.9: 17, an augmented/modified Java virtual machine specific to the inventive Java macros).

b) Limitations at issues "*optimizing the virtual machine based on semantics of the application to be run on the virtual machine, with at least a portion of the semantically enriched opcodes being specific to the application*" (Remark, page 12).

Because Sokolov discloses "*augmenting a bytecode set of the virtual machine with semantically enriched opcodes"* as discussed in a) above*, Sokolov also discloses:*

optimizing the virtual machine based on semantics of the application to be run on the virtual machine (e.g., FIG. 5, col.7: 37-67; FIG. 2B-C, col.8: 62 – col.9: 17),

with at least a portion of the semantically enriched opcodes being specific to the application (e.g., FIG. 4, col.7: 6-19, block 404-406, each application has different frequently repeated bytecode → block 408, specific macro instructions are generated, i.e., semantically enriched opcodes specific to each application).

c) Limitations at issues "*performing a quantitative trade-off between execution time and memory space to determine effective semantically enriched opcodes*" (Remark, pp 12-13).

Examiner notes that Sokolov has not been relied upon to reject particular limitations as argued by Applicants. In an analogous art, Egashira further discloses *performing a quantitative trade-off between execution time and memory space to determine effective semantically enriched opcodes and encoding the semantically enriched opcodes into interpreter action codes based upon the trade-off* (e.g.,

FIG. 3, col.7:60 – col.8: 53, trade-off between code size and execution time;

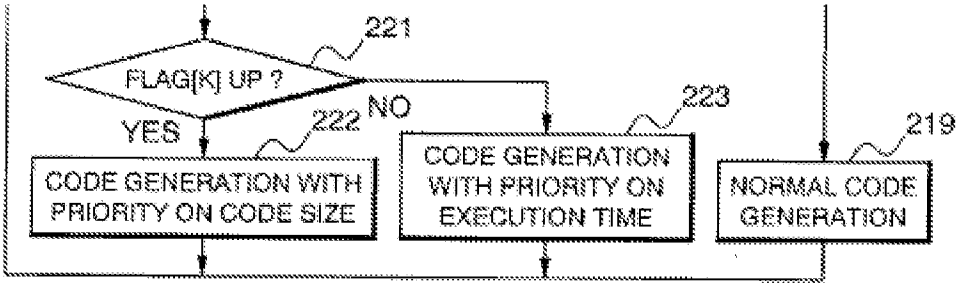

FIG. 3

FIG. 7, col.12: 57 – col.13: 18, prioritizing code size (517) or prioritizing execution time (516) → trade-off between them

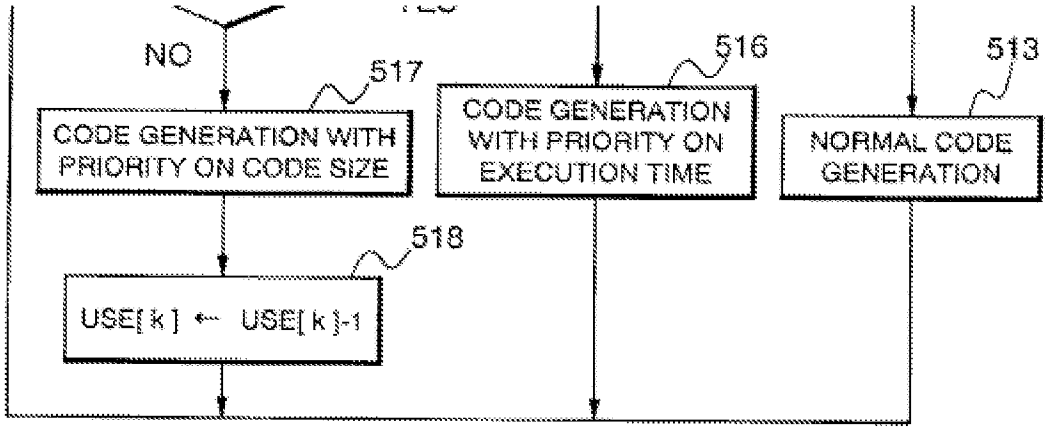

FIG. 7

).

d) Limitations at issues "*optimizing the translation by the interpreter action codes of the semantically enriched opcodes according to a system state, said system state being represented by at least one symbolic variable*" (Remark, page 13).

As an initial matter, examiner notes that the claimed limitation "*a symbolic state…represented by…one symbolic variable*" does not exclude a symbolic

variable/threshold state (such as “a predetermined number of times”) to check against a number of times a sequence of bytecode frequently executes (such as a counter).

Sokolov discloses:

optimizing a translation by the interpreter action codes of the semantically enriched opcodes according to a system state (e.g., FIG. 4, block 406 “Sequence has been counted for at least a predetermined number of times? YES → block 408 “Generate a Java macro instruction that represents the sequence of Java Bytecode instructions”, i.e., only generates macro instruction and passes it to Java interpreter based on whether a counter reaches/exceeds a predefined number of times (a predefined system state),

said system state being represented by at least one symbolic variable (e.g., FIG. 4, blocks 404 and 406, said predefined number of times (said predefined system state) is a variable and compared with a counter – see block 404 “Count the number of times …”, emphasis added).

e) Limitations at issues "*performing a quantitative trade-off between execution time and memory space to determine effective semantically enriched opcodes*" (Remark, pp 13-14).

As discussed in c) above, Egashira further discloses *performing a quantitative trade-off between execution time and memory space to determine effective semantically enriched opcodes and encoding the semantically enriched opcodes into interpreter action codes based upon the trade-off* (e.g.,

FIG. 3, col.7:60 – col.8: 53, trade-off between code size and execution time;

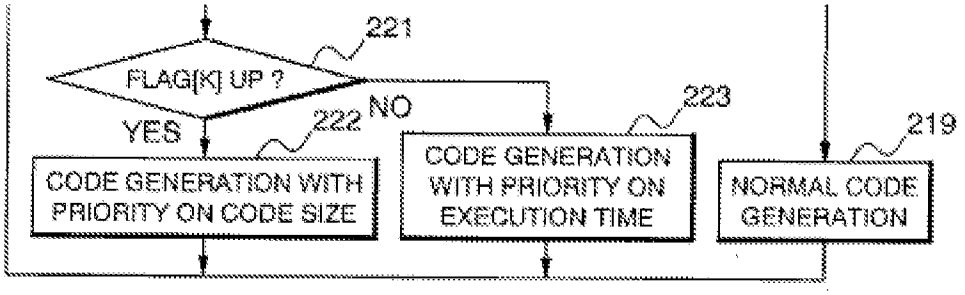

FIG. 3

FIG. 7, col.12: 57 – col.13: 18, prioritizing code size (517) or prioritizing execution time (516) → trade-off between them

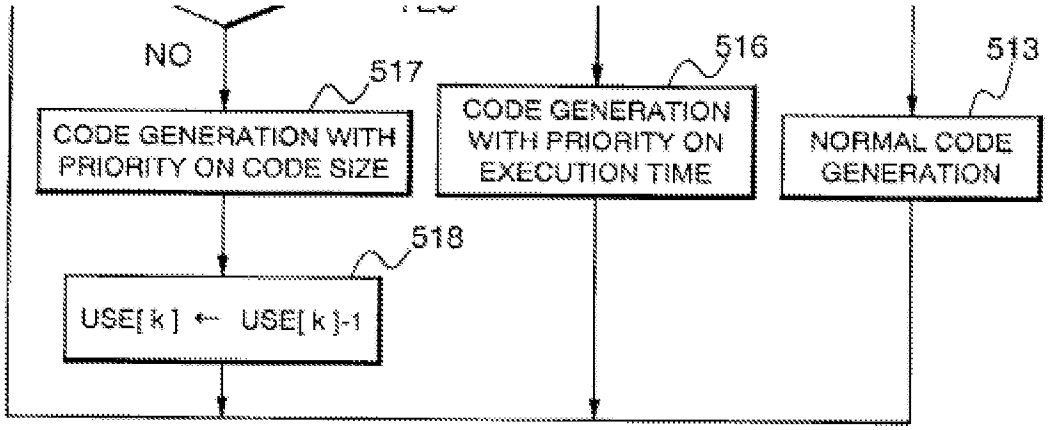

FIG. 7

).

Claim 30 (pp. 15-16):

As discussed above in claim 1, Sokolov in view of Egashira also disclose the claimed limitations in claim 30.

Claims 21 and 31 (page 17):

Examiner notes that the claim language defines "static optimization" in claim 21, lines 2-4: "*the static optimization comprising parsing the application code to identify at least one repeated sequence of bytecodes"* (emphasis added).

Per the plain language of the claims, Sokolov discloses *analyzing an application code using static optimization, the static optimization comprising parsing the application code to identify at least one repeated sequence of bytecodes* (e.g., FIG. 4, block 402 "Read a stream of Java Bytecode instructions …" → block 404 "Count the number of times …", col.2: 19 – col.3: 47, i.e., parsing and identifying at least one repeated sequence of instructions) *and*

replace the at least one repeated sequence of bytecodes with a semantically enriched opcode (e.g., col.5: 28 – col.6: 38).

Claims 23 and 33 (pp 17-18):

Examiner notes that the claimed limitation "a symbolic state" does not exclude a symbolic value/threshold (such as "a predetermined number of times") to check against a number of times a sequence of bytecode frequently executes. Per the plain language of the claims, Sokolov discloses:

discovering at least one repetitive computational sequence used in a symbolic state (e.g., FIG. 4, block 404, col.8: 1-11, while not end of stream (while still in loop), counting the number of times and checking whether it reaches/exceeds a predetermined number of times (a symbolic state))*; and*

generating a semantically enriched opcode corresponding to the at least one repetitive computational sequence used in the symbolic state (e.g., FIG. 4, block 408, col.11: 9-32, "Generate a Java macro instruction that represents the sequence of Java Bytecode instructions").

Claims 24 and 34 (page 19):

Sokolov discloses *the symbolic state comprises at least one of: control flow, data flow, entry points, and operational arguments* (e.g., FIG. 4, block 406 reaching a

predetermined number of times? YES/NO (control flow selected between 2 branches), col.5: 55 – col.6: 26).

Claims 25 and 35 (pp 20-21):

Examiner notes that FIG. 9A presents a new macro instruction is generated from three conventional Java bytecode, which is actually native to Java Virtual Machine 107 (but not other JVM). As Applicants argued, JVM 107 executes said new macro instruction. However, said new macro instruction still reads on the claimed limitation "native code output by said virtual machine" (i.e., it is the JVM 107 generates/outputs the new macro instruction based on a profiling step as illustrated in FIG. 4).

Sokolov discloses *optimizing native code output by the virtual machine using a global optimizing compiler* (e.g., FIG. 4, col.7: 37-67, each application has different frequently repeated Java bytecode stream → block 408, different Java macro instructions are generated; FIG. 1A, col.8: 62 – col.9: 17, a compiler can be used with different Java macro instructions in different applications, i.e., a (global) optimizer compiler for different optimization (locally) performed in each application).

Claims 27, 29, 37, and 39 (pp 21-22):

Sokolov discloses *the method of claim 1, further comprising optimizing the virtual machine by online modification of a generic virtual machine by inserting a stub* (e.g., col.7: 37-67, col.8: 22-29, FIG. 6B, macro instruction generator 602),

the stub automatically loading a semantically enriched opcode when the virtual machine encounters an identified code segment with a bytestream (e.g., col.6: 66 – col.7: 19; col.8: 29-32, said macro instruction generator 602 ("the stub") receives/loads inventive Java instruction 630 ("a semantically enriched opcode") to generate "New-Dup" 624).

Claim Rejections – 35 USC §103

6. The following is a quotation of 35 U.S.C. 103(a) which forms the basis for all obviousness rejections set forth in this Office action:

> (a) A patent may not be obtained though the invention is not identically disclosed or described as set forth in section 102 of this title, if the differences between the subject matter sought to be patented and the prior art are such that the subject matter as a whole would have been obvious at the time the invention was made to a person having ordinary skill in the art to which said subject matter pertains. Patentability shall not be negatived by the manner in which the invention was made.

7. Claims 1 and 21-39 are rejected under 35 U.S.C. 103(a) as being unpatentable over Sokolov (art of record, US Patent No. 6,988,261) in view of Egashira (art of record, US Patent No. 6,014,519).

Claim 1:

Sokolov discloses *a method of optimizing the performance of an interpreter based runtime system, the runtime system including a virtual machine, the virtual machine adapted to run an application in the context of the runtime environment, the method comprising:*

augmenting a bytecode set of the virtual machine with semantically enriched opcodes (e.g., FIG. 12A, conventional Java bytecode "Getfield" and "Astorex" are augmented/replaced by inventive opcode "Get_Store" (a semantically enriched opcode); FIG. 13C, conventional Java bytecode "lastore" and "dastore" are augmented/replaced by another inventive opcode "ASTOREL" (another semantically enriched opcode"; FIG. 2B, col.6: 55 – col.6: 26).

thereby constituting an application domain-specific virtual machine (e.g., col.4: 3-15, col.6: 23-26, an augmented/modified Java virtual machine specific to the inventive opcodes, col.7: 51-59; FIG. 2A-B, col.8: 6 - col.9: 17, an augmented/modified Java virtual machine specific to the inventive Java macros);

optimizing the virtual machine based on semantics of the application to be run on the virtual machine (e.g., FIG. 5, col.7: 37-67; FIG. 2B-C, col.8: 62 – col.9: 17),

with at least a portion of the semantically enriched opcodes being specific to the application (e.g., FIG. 4, col.7: 6-19, block 404-406, each application has different frequently repeated bytecode → block 408, specific macro instructions are generated, i.e., semantically enriched opcodes specific to each application)*;*

performing a quantitative comparison between execution time and memory space (e.g., col.2: 64 – col.3: 9 and col.5: 66 – col.6: 9, the inventive opcodes executes faster (fewer instructions) and occupy less memory space (fewer instructions); col.8: 1-11)

to determine effective semantically enriched opcodes and encoding the semantically enriched opcodes into interpreter action codes based upon the comparison (e.g., col.3: 14-col.4: 15 and FIG. 4, after reaching a predefined threshold, generating Java macro instructions (inventive opcodes/semantically enriched opcodes) based on the comparison (executing faster and less memory space); col.5: 28 – col.6: 38; col.11: 9-32)*;*

analyzing frequently executed bytecodes (e.g., FIG. 4, col.6: 66 – col.7: 19) *and*

encoding the semantically enriched opcodes into interpreter action codes of the instruction set of the virtual machine to efficiently decode the frequently executed bytecodes (e.g., FIG. 9B, col.9: 50 – col.10: 11)*;*

optimizing the encoding the semantically enriched opcodes into the interpreter action codes according to a system state, said system state being represented by at least one symbolic variable (e.g., FIG. 4, block 406 "Sequence has been counted for at least a predetermined number of times? YES → block 408 "Generate a Java macro instruction that represents the sequence of Java Bytecode instructions", i.e., only generates macro instruction and passes it to Java interpreter based on whether a counter reaches/exceeds a predefined threshold (a predefined system state), wherein said predefined threshold (said predefined system state) is a variable and compared with a counter – see block 404 "Count the number of times …", emphasis added)*; and*

statically embedding the semantically enriched opcode to optimize execution of the interpreter-based runtime system (e.g., FIG. 4, block 408, col.6: 66 – col.7: 19; FIG. 5, col.7: 37-67).

Sokolov does not explicitly disclose *performing a quantitative trade-off between execution time and memory space to determine effective semantically enriched opcodes and encoding the semantically enriched opcodes into interpreter action codes based upon the trade-off.*

However, in an analogous art, Egashira further discloses *performing a quantitative trade-off between execution time and memory space to determine effective semantically enriched opcodes and encoding the semantically enriched opcodes into interpreter action codes based upon the trade-off* (e.g., FIG. 3, col.7:60 – col.8: 53; FIG. 7, col.12: 57 – col.13: 18; col.6: 60 – col.7: 33).

It would have been obvious to a person having ordinary skill in the art at the time the invention was made to combine Egashira's teaching into Sokolov' teaching. One would have been motivated to do so to generate an object module file with a shorter execution time in a range of code size as suggested by Egashira (e.g., col.2: 45-54; col.3: 22-29; col.3: 62 – col.4: 39).

Claim 21:

Sokolov discloses *the method of claim 1, further comprising analyzing an application code using static optimization, the static optimization comprising parsing the application code to identify at least one repeated sequence of bytecodes* (e.g., FIG. 4, block 402 "Read a stream of Java Bytecode instructions …" → block 404 "Count the number of times …", col.2: 19 – col.3: 47, i.e., parsing and identifying at least one repeated sequence) *and*

replace the at least one repeated sequence of bytecodes with a semantically enriched opcode (e.g., col.5: 28 – col.6: 38).

Claim 22:

Sokolov discloses *the method of claim 1, further comprising analyzing an application code using dynamic optimization, the dynamic optimization comprising: analyzing temporal behavior of the application code during execution* (e.g., col.8: 12-46)*; and*

identifying and replacing at least one repeated computational sequence in a bytecode stream with a semantically enriched opcode (e.g., col.9: 23 – col.10: 34).

Claim 23:

Sokolov discloses *the method of claim 1, further comprising: discovering at least one repetitive computational sequence used in a symbolic state* (e.g., FIG. 4, block 404, col.8: 1-11, while not end of stream (while still in loop), counting the number of times and checking whether it reaches/exceeds a predetermined number of times (a symbolic state))*; and*

generating a semantically enriched opcode corresponding to the at least one repetitive computational sequence used in the symbolic state (e.g., FIG. 4, block 408, col.11: 9-32, "Generate a Java macro instruction that represents the sequence of Java Bytecode instructions").

Claim 24:

Sokolov discloses *the method of claim 23, wherein the symbolic state comprises at least one of: control flow, data flow, entry points, and operational arguments* (e.g., FIG. 4, block 406 reaching a predetermined number of times? YES/NO (control flow selected between 2 branches), col.5: 55 – col.6: 26).

Claim 25:

Sokolov discloses *the method of claim 1, further comprising optimizing native code output by the virtual machine using a global optimizing compiler* (e.g., FIG. 4, col.7: 37-67, each application has different frequently repeated Java bytecode stream → block 408, different Java macro instructions are generated; FIG. 1A, col.8: 62 – col.9: 17, a compiler can be used with different Java macro instructions in different applications, i.e., a global optimizer compiler).

Claim 26:

Sokolov discloses *the method of claim 1, further comprising optimizing the virtual machine by offline compilation of the virtual machine by a host device* (e.g., col.3: 14 – col.4: 15; col.6: 66 – col.7: 19).

Claim 27:

Sokolov discloses *the method of claim 1, further comprising optimizing the virtual machine by online modification of a generic virtual machine by inserting a stub* (e.g., col.2: 64 – col.3: 9; col.7: 37-67),

the stub automatically loading a semantically enriched opcode when the virtual machine encounters an identified code segment with a bytestream (e.g., col.6: 66 – col.7: 19).

Claim 28:

Sokolov discloses *the method of claim 1, further comprising offline embedding of the semantically enriched opcodes in an application by substituting a semantically enriched opcode for a corresponding code segment* (e.g., col.9: 50 - col.10: 11).

Claim 29:

Sokolov discloses *the method of claim 1, further comprising online embedding of semantically enriched opcodes by a class loader* (e.g., col.2: 19 – col.3: 47),

said class loader substituting a semantically enriched opcode for a corresponding code segment (e.g., col.3: 14 – col.4: 15).

Claim 30:

Sokolov discloses *a system for optimizing performance of an interpreter based runtime system, the runtime system including a virtual machine, the system comprising:*

application code; an embedded processor; a virtual machine configured to translate said application code into native machine code compatible with said embedded processor (e.g., col.2: 19 – col.3: 47; col.3: 14 – col.4: 15);

a detection module, said detection module being configured to analyze said application code to identify code segments that could be efficiently represented as semantically enriched opcodes (e.g., FIG. 2B, col.5: 55 – col.6: 26; FIG. 4, col.6: 66 - col.7: 19)*;*

at least a portion of the semantically enriched opcodes being specific to said application (e.g., FIG. 4, col.7: 6-19, block 404-406, each application has different frequently repeated bytecode → block 408, specific macro instructions are generated, i.e., specific semantically enriched opcodes)*;*

an embedding module, said embedding module being configured to embed said semantically enriched opcodes in said application (e.g., col.5: 66 – col.6: 9; col.8: 12-46; col.9: 23 – col.10: 34)*;*

a code generation module, said code generation module being configured to generate optimized action code for translating said semantically enriched opcodes according to symbolic states (e.g., FIG. 4, col.6: 66 – col.7: 19; FIG. 6A, "New" and "Dup"; FIG. 9A, block 902 and “Loop 1”; FIG. 9B, blocks 910 and 920; col.11: 9-32, Appendix A),

each of said symbolic states being represented by at least one symbolic variable (e.g., FIG. 4, block 406 “Sequence has been counted for at least a predetermined number of times? YES → block 408 “Generate a Java macro instruction that represents the sequence of Java Bytecode instructions”, i.e., only generates macro instruction and passes it to Java interpreter based on whether a counter reaches/exceeds a predefined threshold (a predefined system state), wherein said predefined threshold (said predefined system state) is a variable and compared with a counter – see block 404 “Count the number of times …”, emphasis added)*; and*

a build module configured create an application domain-specific virtual machine by incorporating said optimized action code and a bytecode set comprising said semantically enriched opcodes into said virtual machine (e.g., FIG. 5, col.7: 37-67; FIG. 7B-C, col.8: 62 - col.9: 17)*.*

Sokolov does not explicitly disclose *performing a quantitative trade-off between execution time and memory space to determine effective semantically enriched opcodes and encoding the semantically enriched opcodes into interpreter action codes based upon the trade-off.*

However, in an analogous art, Egashira further discloses *performing a quantitative trade-off between execution time and memory space to determine effective semantically enriched opcodes and encoding the semantically enriched opcodes into interpreter action codes based upon the trade-off* (e.g., FIG. 3, col.7:60 – col.8: 53; FIG. 7, col.12: 57 – col.13: 18; col.6: 60 – col.7: 33).

It would have been obvious to a person having ordinary skill in the art at the time the invention was made to combine Egashira's teaching into Sokolov' teaching. One would have been motivated to do so to generate an object module file with a shorter execution time in a range of code size as suggested by Egashira (e.g., col.2: 45-54; col.3: 22-29; col.3: 62 – col.4: 39).

Claim 31:

Sokolov discloses *the system of claim 30, wherein said detection module is configured to analyze said application code using static optimization, said static optimization comprising parsing said application code to identify at least one repeated sequence of bytecodes* (e.g., col.5: 28 – col.6: 38; col.9: 23 – col.10: 34) *and*

replace said at least one repeated sequence of bytecodes with a semantically enriched opcode (e.g., col.5: 66 – col.6: 9; col.7: 37-67).

Claim 32:

Sokolov discloses *the system of claim 31, wherein said detection module is further configured to analyze said application code using dynamic optimization, said dynamic optimization comprising: analyzing temporal behavior of the application code during execution* (e.g., col.2: 64 – col.3: 9)*; and*

identifying and replacing at least one repeated computational sequence in a bytecode stream with a semantically enriched opcode (e.g., col.3: 14 – col.4: 15; col.5: 66 – col.6: 9).

Claim 33:

Sokolov discloses *the system of claim 30, wherein said generation module is further configured to: discover at least one repetitive computational sequence used in a said symbolic state* (e.g., col.8: 12-46)*; and*

generate a semantically enriched opcode corresponding to the at least one repetitive computational sequence used within said symbolic state (e.g., col.9: 23 – col.10: 34).

Claim 34:

Sokolov discloses *the system of claim 33, wherein said symbolic state comprises at least one of: control flow, data flow, entry points, and operational arguments* (e.g., col.5: 66 – col.6: 9; col.8: 1-11).

Claim 35:

Sokolov discloses *the system of claim 30, further comprising a global optimizer compiler configured to optimizing native code output by said virtual machine* (e.g., col.5: 28 - col.6: 38).

Claim 36:

Sokolov discloses *the system of claim 30, wherein said virtual machine is compiled offline by a host device* (e.g., col.8: 12-46).

Claim 37:

Sokolov discloses *the system of claim 30, wherein said virtual machine is modified online by inserting a stub, said stub automatically loading a semantically*

enriched opcode when said virtual machine encounters an identified code segment with a bytestream (e.g., col.7: 37-67).

Claim 38:

Sokolov discloses *the system of claim 30, wherein said application code is modified offline by substituting a semantically enriched opcode for a corresponding code segment contained with said application code* (e.g., col.3: 14 – col.4: 15; col.6: 66 – col.7: 19).

Claim 39:

Sokolov discloses *the system of claim 30, wherein a class loader embeds said semantically enriched opcodes online, said class loader substituting a semantically enriched opcodes for a corresponding code segment* (e.g., col.2: 19 – col.3: 47; col.5: 28 – col.6: 38).

Conclusion

8. **THIS ACTION IS MADE FINAL**. See MPEP § 706.07(a). Applicant is reminded of the extension of time policy as set forth in 37 CFR 1.136(a).

A shortened statutory period for reply to this final action is set to expire THREE MONTHS from the mailing date of this action. In the event a first reply is filed within TWO MONTHS of the mailing date of this final action and the advisory action is not mailed until after the end of the THREE-MONTH shortened statutory period, then the shortened statutory period will expire on the date the advisory action is mailed, and any extension fee pursuant to 37 CFR 1.136(a) will be calculated from the mailing date of the advisory action. In no event, however, will the statutory period for reply expire later than SIX MONTHS from the date of this final action.

9. Any inquiry concerning this communication should be directed to examiner Thuy (Twee) Dao, whose telephone/fax numbers are (571) 272 8570 and (571) 273 8570,

respectively. The examiner can normally be reached on every Tuesday, Thursday, and Friday from 6:00AM to 6:00PM.

If attempts to reach the examiner by telephone are unsuccessful, the examiner's supervisor, Tuan Q. Dam, can be reached at (571) 272 3695.

Any inquiry of a general nature of relating to the status of this application or proceeding should be directed to the TC 2100 Group receptionist whose telephone number is (571) 272 2100.

Information regarding the status of an application may be obtained from the Patent Application Information Retrieval (PAIR) system. Status information for published applications may be obtained from either Private PAIR or Public PAIR. Status information for unpublished applications is available through Private PAIR only. For more information about the PAIR system, see http://pair-direct.uspto.gov. Should you have questions on access to the Private PAIR system, contact the Electronic Business Center (EBC) at 866-217-9197 (toll-free).

/Twee Dao/
Examiner, Art Unit 2192

/Tuan Q. Dam/
Supervisory Patent Examiner, Art Unit 2192